AMENDMENTS TO THE CLAIMS

Claims 1-48 (canceled).

49. (New) A method of compiling a source code written in an object-oriented source language into executable machine-language code for a target computer, comprising:

defining in said source code, using said source language, software-visible hardware objects of said target computer as pre-existing objects of classes defined in said source language, wherein at least one of member methods of said classes contain statements in said source language, said statements containing expressions evaluating to values of operation codes and associated operand specifiers in said machine-language; and

compiling results of said defining into said machine-language code by expanding, inline, every invocation of said member methods of said classes to said values of operation codes and associated operand specifiers to which their expressions evaluate, said expanding inline of invocations continuing recursively until all invocations of said member methods have been replaced by operation codes and associated operand specifiers, said compiling accomplished without requiring application of an intermediate language.

50. (New) The method of compiling of claim 49, further comprising:

defining in said source code, using said source language, at least one subclass of a subroutine class, each subclass implementing a method of invoking, out-of-line, a subroutine of said subroutine class by defining a call member method, wherein each subclass further defines an instance member holding code of said subroutine, said code of said subroutine preceded by code operable for saving registers, and said code of said subroutine followed by code operable for restoring registers and transferring control to a point immediately after a point from which said subroutine is invoked out-of-line;

compiling one or more member methods defined in said source code by

generating objects of said at least one subclass, said objects of said at least one subclass being subroutines invoked out-of-line; and

compiling results of said defining at least one subclass by expanding at least one invocation, contained in said results of said defining at least one subclass, of said member methods of said classes to be inline expansions of said call member method, wherein each subroutine to be invoked out-of-line is compiled as an object of said subclass of a corresponding call member method.

51. (New) The method of compiling of claim 50, further comprising:

defining in said source code, using said source language:

at least one of hardware and software objects involved in each subroutine being invoked out-of-line; and

at least one of objects carrying information into or out of each subroutine being invoked out-of-line and objects that are allocated or deallocated by each subroutine being invoked out-of-line; and

evaluating use of said at least one of hardware and software objects across subroutine invocations indicated in said source code and, based upon results of said evaluating, generating code for allocating, deallocating, saving, and restoring corresponding hardware and software objects.

52. (New) The method of compiling of claim 50, further comprising:

defining in said source code, using said source language, at least one class declared for implementing representations of abstract data types defined in said source code or predefined in said source language;

defining in said source code, using said source language, variables of said abstract data types; and

said compiler generating code to instantiate objects for representing said variables, said objects being instances of classes declared to represent said abstract data types.

53. (New) The method of claim 49, further comprising compiling a source code written in an object-oriented source language into executable machine-language code for a second target computer, the method comprising:

separating results of said defining, comprising:

placing in a first library the classes having member methods containing statements in said source language, the statements having expressions evaluating to values of operation codes and associated operand specifiers in said machine-language for the target computer; and

placing remaining classes in a second library;

defining a third library of source code containing classes equivalent to the classes of the first library, the classes of the third library including member methods containing statements having expressions evaluating to values of operation codes and associated operand specifiers in said machine-language code for the second target computer; and

compiling the second library and the third library into said machine-language code for the second target computer, by the method of compiling of claim 49.

54. (New) A method of compiling literals in a source code, written in an object-oriented source language, into at least one of data objects and executable machine-language code for a target computer, comprising:

defining in said source code, using said source language, classes having initializer member methods that take as arguments an internal representation of said literals of a compiler, said initializer member methods previously compiled from said source code;

upon parsing one of said literals, said compiler calling one of said initializer member methods, said initializer member method taking said internal representation of said parsed literal as an argument resulting in an initialized object; and

said compiler performing at least one of:

incorporating said initialized object into an output of said compiler; and

using said initialized object in subsequent compilation steps.

55. (New) A method of compiling two object-oriented classes from a single class literal in a source code, comprising:

in said source code, labeling at least one member method defined in said single class literal as belonging to a constant version of a class, said constant version of said class differentiated from a variable version of said class; and

said compiler generating two classes from said single class literal, said constant version of said class being a base class of said variable version of said class.

56. (New) A method of compiling two object-oriented class interfaces from a single class interface literal in a source code, comprising:

in said source code, labeling at least one member method defined in said class interface literal as belonging to a constant version of said class interface, said constant version of said class interface differentiated from a variable version of said class interface; and

said compiler generating two class interfaces from said single class interface literal, said constant version of said class interface being abase class of said variable version of said class interface.

57. (New) A storage medium encoded with machine-readable program code for compiling a source code written in an object-oriented source language into executable machine-language code for a target computer, the program code including instructions for causing a computer to implement a method, comprising:

defining in said source code, using said source language, software-visible hardware objects of said target computer as pre-existing objects of classes defined in said source language, wherein at least one of member methods of said classes contain statements in said source language, said statements containing expressions evaluating to values of operation codes and associated operand specifiers in said machine-language; and

compiling results of said defining into said machine-language code by expanding,

inline, every invocation of said member methods of said classes to said values of operation codes and associated operand specifiers to which their expressions evaluate, said expanding inline of invocations continuing recursively until all invocations of said member methods have been replaced by operation codes and associated operand specifiers, said compiling accomplished without requiring application of an intermediate language.

58. (New) The storage medium of claim 57, further comprising instructions for causing the computer to implement:

defining in said source code, using said source language, at least one subclass of a subroutine class, each subclass implementing a method of invoking, out-of-line, a subroutine of said subroutine class by defining a call member method, wherein each subclass further defines an instance member holding code of said subroutine, said code of said subroutine preceded by code operable for saving registers, and said code of said subroutine followed by code operable for restoring registers and transferring control to a point immediately after a point from which said subroutine is invoked out-of-line;

compiling one or more member methods defined in said source code by generating objects of said at least one subclass, said objects of said at least one subclass being subroutines invoked out-of-line; and

compiling results of said defining at least one subclass by expanding at least one invocation, contained in said results of said defining at least one subclass, of said member methods of said classes to be inline expansions of said call member method, wherein each subroutine to be invoked out-of-line is compiled as an object of said subclass of a corresponding call member method.

59. (New) The storage medium of claim 58, further comprising instructions for causing the computer to implement:

defining in said source code, using said source language:

at least one of hardware and software objects involved in each subroutine

being invoked out-of-line; and

at least one of objects carrying information into or out of each subroutine being invoked out-of-line and objects that are allocated or deallocated by each subroutine being invoked out-of-line; and

evaluating use of said at least one of hardware and software objects across subroutine invocations indicated in said source code and, based upon results of said evaluating, generating code for allocating, deallocating, saving, and restoring corresponding hardware and software objects.

60. (New) The storage medium of claim 58, further comprising instructions for causing the computer to implement:

defining in said source code, using said source language, at least one class declared for implementing representations of abstract data types defined in said source code or predefined in said source language;

defining in said source code, using said source language, variables of said abstract data types; and

said compiler generating code to instantiate objects for representing said variables, said objects being instances of classes declared to represent said abstract data types.

61. (New) The storage medium of claim 57, further comprising machine-readable program code for compiling a source code written in an object-oriented source language into executable machine-language code for a second target computer, the program code including instructions for causing the computer to implement:

separating results of said defining, comprising:

placing in a first library the classes having member methods containing statements in said source language, the statements having expressions evaluating to values of operation codes and associated operand specifiers in said machine-language for the target computer; and

placing remaining classes in a second library;

defining a third library of source code containing classes equivalent to the classes of the first library, the classes of the third library including member methods containing statements having expressions evaluating to values of operation codes and associated operand specifiers in said machine-language code for the second target computer; and

compiling the second library and the third library into said machine-language code for the second target computer, by the method of compiling of claim 57.

62. (New) A storage medium encoded with machine-readable program code for compiling literals in a source code, written in an object-oriented source language, into at least one of data objects and executable machine-language code for a target computer, the program code including instructions for causing a computer to implement a method, comprising:

defining in said source code, using said source language, classes having initializer member methods that take as arguments an internal representation of said literals of a compiler, said initializer member methods previously compiled from said source code;

upon parsing one of said literals, said compiler calling one of said initializer member methods, said initializer member method taking said internal representation of said parsed literal as an argument resulting in an initialized object; and

said compiler performing at least one of:

incorporating said initialized object into an output of said compiler; and

using said initialized object in subsequent compilation steps.

63. (New) A storage medium encoded with machine-readable program code for compiling two object-oriented classes from a single class literal in a source code, the program code including instructions for causing a computer to implement a method, comprising:

in said source code, labeling at least one member method defined in said single class literal as belonging to a constant version of a class, said constant version of said class differentiated from a variable version of said class; and

said compiler generating two classes from said single class literal, said constant version of said class being a base class of said variable version of said class.

64. (New) A storage medium encoded with machine-readable program code for compiling two object-oriented class interfaces from a single class interface literal in a source code, the program code including instructions for causing a computer to implement a method, comprising:

in said source code, labeling at least one member method defined in said class interface literal as belonging to a constant version of said class interface, said constant version of said class interface differentiated from a variable version of said class interface; and

said compiler generating two class interfaces from said single class interface literal, said constant version of said class interface being abase class of said variable version of said class interface.

65. (New) A signal propagated over a propagation medium, the signal encoded with code for compiling a source code written in an object-oriented source language into executable machine-language code for a target computer, the code including instructions for allowing a computer to implement a method, comprising:

defining in said source code, using said source language, software-visible hardware objects of said target computer as pre-existing objects of classes defined in said source language, wherein at least one of member methods of said classes contain statements in said source language, said statements containing expressions evaluating to values of operation codes and associated operand specifiers in said machine-language; and

compiling results of said defining into said machine-language code by expanding, inline, every invocation of said member methods of said classes to said values of operation codes and associated operand specifiers to which their expressions evaluate, said expanding inline of invocations continuing recursively until all invocations of said

member methods have been replaced by operation codes and associated operand specifiers, said compiling accomplished without requiring application of an intermediate language.

66. (New) The signal propagated over the propagation medium of claim 65, wherein the method further comprises:

defining in said source code, using said source language, at least one subclass of a subroutine class, each subclass implementing a method of invoking, out-of-line, a subroutine of said subroutine class by defining a call member method, wherein each subclass further defines an instance member holding code of said subroutine, said code of said subroutine preceded by code operable for saving registers, and said code of said subroutine followed by code operable for restoring registers and transferring control to a point immediately after a point from which said subroutine is invoked out-of-line;

compiling one or more member methods defined in said source code by generating objects of said at least one subclass, said objects of said at least one subclass being subroutines invoked out-of-line; and

compiling results of said defining at least one subclass by expanding at least one invocation, contained in said results of said defining at least one subclass, of said member methods of said classes to be inline expansions of said call member method, wherein each subroutine to be invoked out-of-line is compiled as an object of said subclass of a corresponding call member method.

67. (New) The signal propagated over the propagation medium of claim 66, wherein the method further comprises:

defining in said source code, using said source language:

at least one of hardware and software objects involved in each subroutine being invoked out-of-line; and

at least one of objects carrying information into or out of each subroutine being invoked out-of-line and objects that are allocated or deallocated by each subroutine

being invoked out-of-line; and

evaluating use of said at least one of hardware and software objects across subroutine invocations indicated in said source code and, based upon results of said evaluating, generating code for allocating, deallocating, saving, and restoring corresponding hardware and software objects.

68. (New) The signal propagated over the propagation medium of claim 66, wherein the method further comprises:

defining in said source code, using said source language, at least one class declared for implementing representations of abstract data types defined in said source code or predefined in said source language;

defining in said source code, using said source language, variables of said abstract data types; and

said compiler generating code to instantiate objects for representing said variables, said objects being instances of classes declared to represent said abstract data types.

69. (New) The signal propagated over the propagation medium of claim 65, wherein the method further comprises compiling a source code written in an object-oriented source language into executable machine-language code for a second target computer, comprising:

separating results of said defining, comprising:

placing in a first library the classes having member methods containing statements in said source language, the statements having expressions evaluating to values of operation codes and associated operand specifiers in said machine-language for the target computer; and

placing remaining classes in a second library;

defining a third library of source code containing classes equivalent to the classes of the first library, the classes of the third library including member methods containing statements having expressions evaluating to values of operation codes and associated

operand specifiers in said machine-language code for the second target computer; and

compiling the second library and the third library into said machine-language code for the second target computer, by the method of compiling of claim 65.

70. (New) A signal propagated over a propagation medium, the signal encoded with code for compiling literals in a source code, written in an object-oriented source language, into at least one of data objects and executable machine-language code for a target computer, the code including instructions for allowing a computer to implement a method, comprising:

defining in said source code, using said source language, classes having initializer member methods that take as arguments an internal representation of said literals of a compiler, said initializer member methods previously compiled from said source code;

upon parsing one of said literals, said compiler calling one of said initializer member methods, said initializer member method taking said internal representation of said parsed literal as an argument resulting in an initialized object; and

said compiler performing at least one of:

incorporating said initialized object into an output of said compiler; and

using said initialized object in subsequent compilation steps.

71. (New) A signal propagated over a propagation medium, the signal encoded with code for compiling two object-oriented classes from a single class literal in a source code, the code including instructions for allowing a computer to implement a method, comprising:

in said source code, labeling at least one member method defined in said single class literal as belonging to a constant version of a class, said constant version of said class differentiated from a variable version of said class; and

said compiler generating two classes from said single class literal, said constant version of said class being a base class of said variable version of said class.

72. (New) A signal propagated over a propagation medium, the signal encoded with code for compiling two object-oriented class interfaces from a single class interface literal in a source code, the code including instructions for allowing a computer to implement a method, comprising:

in said source code, labeling at least one member method defined in said class interface literal as belonging to a constant version of said class interface, said constant version of said class interface differentiated from a variable version of said class interface; and

said compiler generating two class interfaces from said single class interface literal, said constant version of said class interface being abase class of said variable version of said class interface.